ETG -17-62
ETT -17-62

# ２乗則を用いたデジタル乗算器アーキテクチャ

佐々木　秀*　　孫 逸菲　　姚 丹　　小林 春夫（群馬大学大学院）

# Study on Digital Multiplier Architecture Using Square Law

Shu Sasaki†　Yifei Sun,　Dan Yao,　Haruo Kobayashi (Gunma University)

**あらまし**　デジタル乗算器はデジタル計算機、DSP チップ等に広く用いられている。直接的に実現すると全加算器の２次元配列になり、ビット数が増えると回路規模・消費電力・演算時間が大きくなってしまう。この論文ではこの小規模回路実現のため入力 A, B に対し　その和と 2 乗から積 AB を得る式に基づき、デジタル乗算器を RTL レベルで設計・検証した。２乗計算回路を論理回路および分割統治法(Divide & Conquer) 使用の LUT で設計した。

**Abstract**　A digital multiplier is widely used for digital computers and DSP chips. When it is rerealized directly, a two-dimensional array of full adders is required;　as the number of bits increases, its circuit size and power become large and its computation time is also increase. In this paper, the digital multiplier has been designed at the RTL level based on the equations for obtaining the product AB from the sum and square of the inputs A and B. This eables its small scale circuit implementation. We have investigated the squaring calculation circuit with LUT using divide &conquer method and also direct squaring calculation logic.

**キーワード**：デジタル乗算器,　２乗則, デジタル回路,　FPGA
(Keywords, Digital Multiplier, Square Law, Digital Circuit, FPGA)

## 1.　研究背景と目的

デジタル乗算器はデジタル計算機、DSP チップ等に広く用いられている。2 進数の乗算は、2 進数の加算を繰り返しで計算するため、多くの計算量を必要とする。デジタル乗算器を直接的に実現すると全加算器の２次元配列になり、回路規模・消費電力・演算時間が大きくなる問題がある。そのため回路量・消費電力の減少および高速化のために様々なアルゴリズムが提案され、それに基づきデジタル乗算器が設計・実現されてきた。

特にデジタル通信システムでは複雑・大量のデジタル演算をリアルタイムに行う必要があり、小規模デジタル乗算器を実現できれば、それらを多数実装・並列演算でき、そのアプリケーションに有効である。

ここでは２つのデジタル入力 A, B に対し、その和と 2 乗から積 AB を計算する下記の式を用いてデジタル乗算器を実現する方式を検討する。

$$AB = \frac{1}{2}\{(A + B)^2 - A^2 - B^2\} \quad \cdots\cdots(1)$$

この 2 乗演算のために２つの方法を検討する
(i)ルックアップテーブル(Look-Up Table: LUT) を用いる。また LUT と加算器、減算器のサイズを小さくするため A, B, A+B のそれぞれの上位ビット、下位ビットに分割して計算量を低減する方式（Divide & Conquer 法）を用いる。
(ii) ２乗計算をする論理式を書き下すと比較的簡単な構成になることを見出した。もう一つの方法として２乗計算を直接論理回路で実現する方式を検討した。

これまで(i) の方式の研究発表をしてきたが、この論文でではあらたに(ii) の方式を検討したので報告する。(ii) の方式で提案アルゴリズムを FPGA で実現・検証をすべく、検討したアルゴリズムで入力 4bit×4bit（出力 8bit）、入力 8bit ×8bit （出力 16bit）、および入力 16bit x 16bit (出力 32bit) の乗算器を設計しレジスタ・トランスファ・レベル（Register Transfer Level: RTL）シミュレーションを行い、正しくデジタル乗算が行われていることを確認する。そして FPGA で実装し、実際に動作可能かどうかを検証する。この提案方式を実現した回路を内部でパイプライン構成にすれば回路全体の高速化が可能であり、小規模・高速のデジタル乗算器を、一つのチップ内に多数配置して大量のデータ処理を行うことが期待できる。なお、ここでは簡単のため A, B は正の数の場合のみを扱っている。

## 2.　検討したデジタル乗算計算アルゴリズム

### 2.1 対数を用いるアルゴリズム

デジタル乗算を容易に計算する方法の一つとして、対数を用いる演算する手法がある。２つのデータ A, B に対して AB の乗算を、加算器と LUT を用いて次のように計算する。

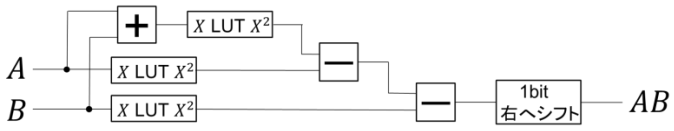
①　A,　B の対数 log A, log B を
　対数データの LUT を用いて得る。
②　log A + log B (= log AB) を
　加算器で得る。

③　log AB から AB を
　　指数データの LUT を用いて得る。

しかし高精度で対数、指数を得るためには LUT のビット数を大きくしなければならず、回路規模が大きくなってしまい、本研究の目的に沿わないため今回は検討対象外とした。

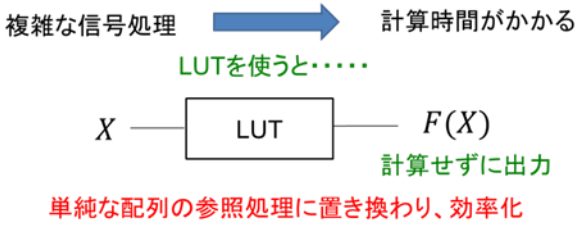
### 2.2 2乗則アルゴリズム (LUT 使用)

検討したアルゴリズムは式（1）の 2 乗則を用いる。1/2 は右シフト演算（実際には結線変更のみでよい）、2 乗の計算部分はルックアップテーブル（Look-Up Table: LUT)または直接 2 乗計算する論理回路を用いることで回路規模削減・低消費電力化・計算時間短縮を実現する。図 1 に式(1)を実現する回路構成を示す。



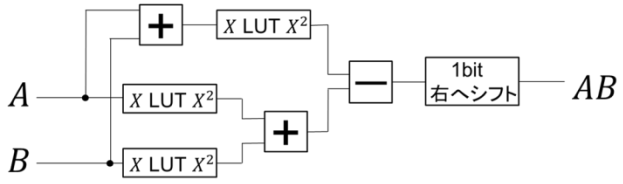

【図 1：2 乗則 式(1) を実現する回路構成】

LUT はメモリ（RAM または ROM）である。入力はメモリの”アドレス”であり、出力はメモリの”データ”である。メモリ内に計算データを記憶しておくことで、LUT で所望の計算結果を得ることができる。

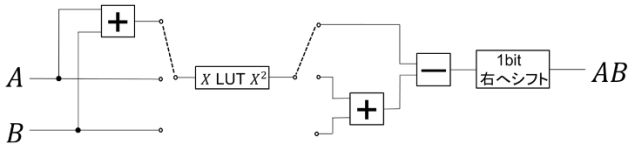



【図 2：ルックアップテーブル(LUT)回路】

図 1 では 3 つの LUT を用いている。演算時間のタイミングを考慮し、図 3 の回路構成も考えられる。他に、1 つの LUT で $A^2, B^2, (A+B)^2$ の計算を行うために逐次的に使うこともできる（図 4)。演算時間は約 3 倍と大きくなってしまうが回路量を約 3 分の 1 に減らすことができる。



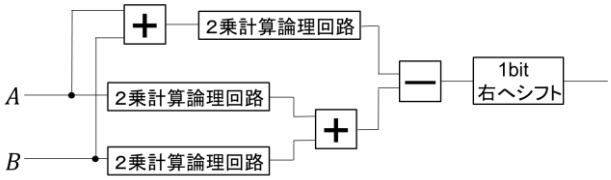

【図 3：演算時間のバランスを考慮した回路】




【図 4：LUT を逐次使用する回路】

### 2.3 2乗則アルゴリズム (2 乗計算論理回路 使用)

前述において LUT を使用して 2 乗計算回路を実現したが、扱う bit 数が大きければ大きいほどその容量を大きくしなければならない。そのため、真理値表（図 6）を用いて 2 乗の結果を出力する専用回路を検討した。




【図 5：2 乗演算論理回路を用いた回路】

ここで『2 乗演算回路』とは例えば 4bit x 4bit で 8bit 出力の場合，図 6 の真理値表を実現する論理回路である。

$$O0 = I0$$
$$O1 = 0$$
$$O2 = I1\bar{I}0$$
$$O3 = (I2 \oplus I1)I0$$
$$O4 = \bar{I}3I2(\bar{I}1 + I0) + I3\bar{I}2I0 + I3I2\bar{I}1\bar{I}0$$
$$O5 = (I3 \oplus I2)I1 + I3I2I0$$
$$O6 = I3\bar{I}2 + I3I2I1$$
$$O7 = I3I2$$

【図 6：出力 8bit の論理式】

多ビットの場合も比較的簡単な論理式で実現できる。

### 2.4 次ステップの検討アルゴリズム

次の 2 乗則アルゴリズムも有力である。

$$AB = \frac{1}{4}\{(A+B)^2 - (A-B)^2\} \quad \cdots\cdots\cdots (2)$$

加減算回数は 3 回と同じであるが、LUT 参照処理は 2 回になる。ここでは簡単のため A, B が正の数の場合のみを扱っているので　式(1)を検討した。次のステップとして式(2)を検討する予定である。

## 3 シミュレーションによる検討

検討アルゴリズム・回路構成の正当性を検証するため「Verilog HDL」を用いて回路シミュレーションを行った。

具体的には、シミュレーションソフト上に回路構成を実現し、二つの入力値を変えて演算させて結果を出力し、結果が正しいかどうかをチェックした。入力が 4bit×4bit で出力が 8bit の場合は、数字の組み合わせが 16×16 = 256

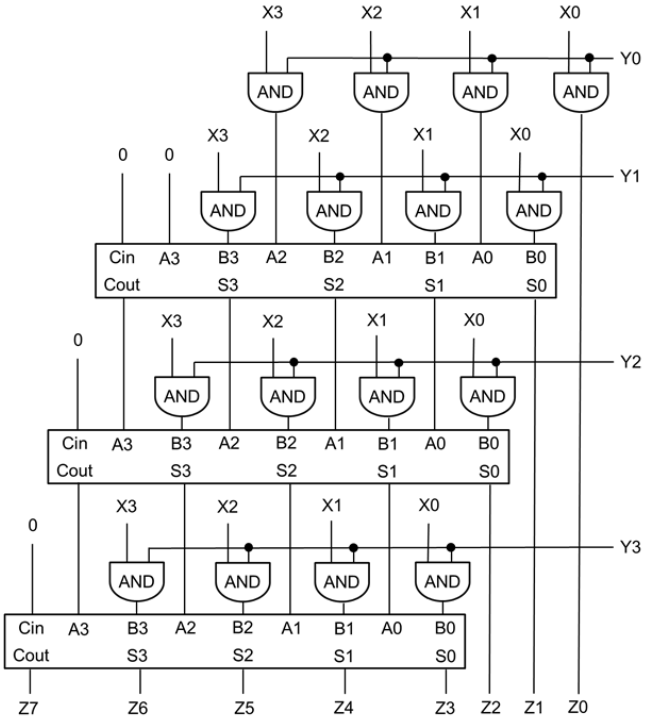
通り、入力が 8bit×8bit で出力が 16bit の場合、256×256 = 65536 通り、入力が 16bit×16bit で出力が 32bit の場合は 65536×65536 = 4294967296 通りある。それらすべての数値で提案アルゴリズムが正しく乗算を行っていることを確認した。

シミュレーション結果を図 12 に記載する。このプログラムを使用すれば提案アルゴリズムの FPGA 実現ができる。今回は【Spartan3E】を用いることで 4bit×4bit の回路を実装し、その動作を確認した。

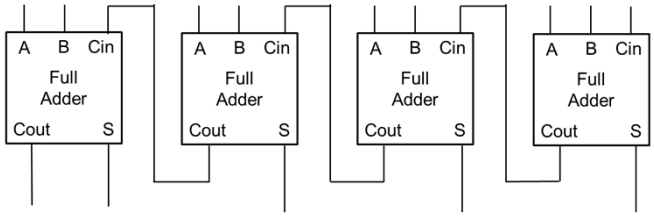
### 3.1 従来乗算器との比較

計算アルゴリズムを用いて乗算器の回路設計を行い、シミュレーション、FPGA 実装を経て正しく動作することを確認した。

ここで、従来の乗算器との比較を行う。最初に述べたとおり、直接的なデジタル乗算器は加算器を連続的に組み合わせて構成されている（図 6-1,6-2 を参照）。

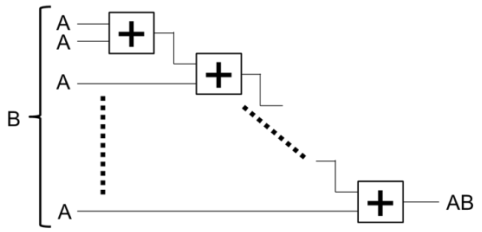



【図 7-1. 従来の乗算器の回路図（4bit）】

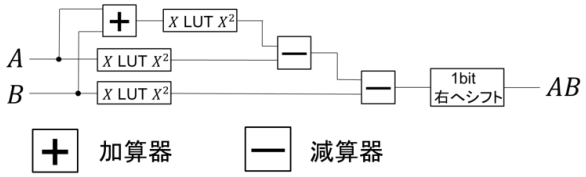



【図 7-2. 全加算器の回路図】

この回路と、理論を元に構成したデジタル乗算回路と RTL レベルで比較する。（図 8,9,10 を参照）




【図 8：従来の乗算器の回路図】




【図 9：構成した回路図（divide&conquer 無し）】

## 4. まとめと今後の課題

二乗則の計算アルゴリズムを用いた回路構成と、そのハードウェア実現を検討し、FPGA 実装のための RTL レベルシミュレーションで動作を確認した。今後は具体的には次を重点的に検証していく。

・従来法との回路量の定量的な比較
・計算精度と演算器・LUT のビット数の明確化
・実装 FPGA 動作クロック周波数）と
　計算速度の明確化
・マイナスの数値の場合の計算。

デジタル乗算器はすべて２進数で表現されており、マイナスがつくと２の補数によってマイナスを表現する。ビット分割の際にはその配慮が必要であるが、今後はそれに対応していく。

・式(2) のアルゴリズムの適用

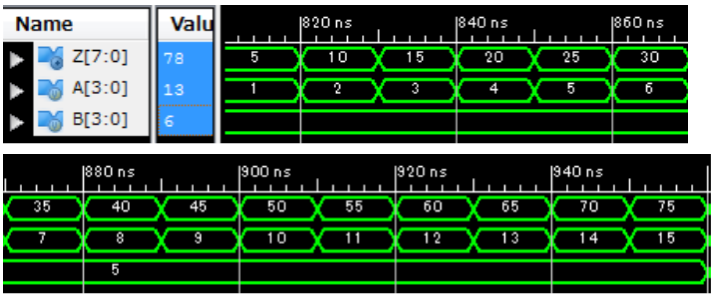
**謝辞：** デジタル乗算器のアルゴリズムに関し有益なコメントをいただきました群馬大学　魏書剛先、大阪工業大学牧野博之先生に感謝いたします。FPGA 実装にアドバイスをいただきました弓仲康史先生、王俊善さん、李从兵さんに感謝いたします。

### 【参考文献】

[1] A. V. Oppenheim, and R. W. Shafer, Digital Signal Processing, Printice-Hall (1975)
[2] L. Cohen, Time-Frequency Analysis, Prentice Hall
[3] A Technical Tutorial on Digital Signal Synthesis, Analog Devices, Inc. (1999).
[4] N. Weste, D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, Addison Wesley (2010)
[5] 佐々木秀、小林春夫「短時間スペクトラム解析の計算アーキテクチャの検討」第 5 回電気学会東京支部栃木・群馬支所合同研究発表会　ETT-15-69, ETG-15-69 宇都宮　(2015 年 3 月)
[6] 佐々木秀、小林春夫「2 乗則を用いたデジタル乗算器ア

ルゴリズムの検討」第 38 回多値論理フォーラム、北海道大学　(2015 年 9 月)
[7] 佐々木秀、小林春夫「2 乗則を用いたデジタル乗算器アルゴリズムと FPGA 実装の検討」8 月度信号処理研究会、千葉大学(2016 年 8 月)

## 付録　提案手法を用いたデジタル乗算器の RTL シミュレーション




【図 10：4bit×4bit のシミュレーション】

8bit×8bit の回路プログラムを入力し、入力値 A , B の値を 10ns、160ns 毎に変化させ、その区間における演算結果を波形上に表示した。

演算結果の確認として図 11 中の A , B , Z のそれぞれには処理することのできる最大 bit 数を示している。この bit 数は入力の bit 数によって変化し、入力値の bit 数の倍以上の容量が必要になる。

Value における数値はシミュレーション結果におけるカーソルの位置の値を表示している。

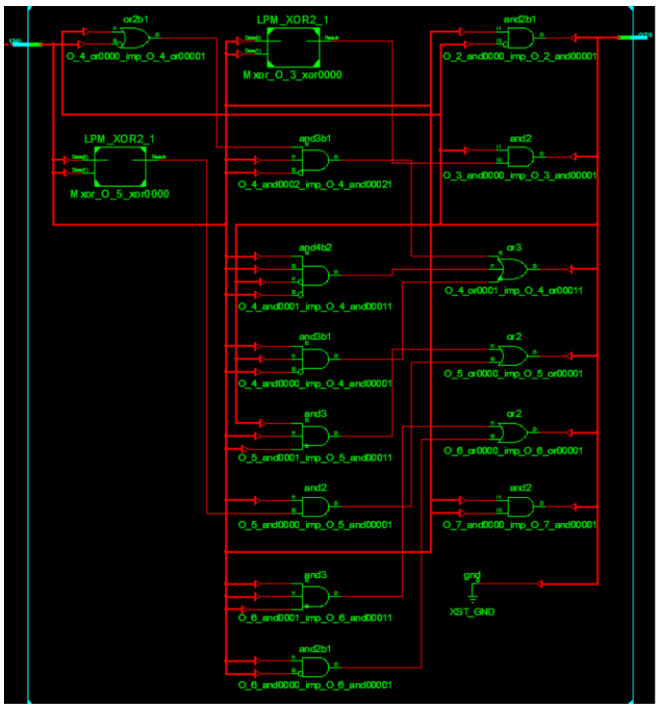
A = 6
B = 13
Z = 78

このため、Value ではその値が表示されている。
また、これらの計算はすべて 2 進数で行っているが、わかりやすくするため結果を 10 進数で表示した。

図 11 中に示したとおり、それぞれの入力値 A , B に対して出力 Z は A×B の値を導いている。これで検討したアルゴリズムが回路上に反映できることが示せた。また他の検討方式におけるシミュレーションでも同様の結果を得ることが出来たため、今回は割愛する。

| | I3 | I2 | I1 | I0 | | O7 | O6 | O5 | O4 | O3 | O2 | O1 | O0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 9 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 16 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 25 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 | 36 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 49 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 8 | 1 | 0 | 0 | 0 | 64 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 81 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 10 | 1 | 0 | 1 | 0 | 100 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 11 | 1 | 0 | 1 | 1 | 121 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 12 | 1 | 1 | 0 | 0 | 144 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 169 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 14 | 1 | 1 | 1 | 0 | 196 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 225 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

【図 11：入力 4bit、出力 8bit の真理値表】




【図 12：２乗計算論理回路図】